Part 1: Implement Logistic Regression

Index

List of Figures

1] Overview:

Goal of the project is to use Gradient Descent for logistic regression to train a model using a group of hyper-parameters to classify whether a patient has diabetes(class 1) or not (class 0). Based on Pima Indians diabetes dataset.

2] Dataset Description & Analysis:

Pima Indians diabetes dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and can be used to predict whether a patient has diabetes based on certain diagnostic factors. Starting off, I use Python 3.3 to implement the model. The dataset has only 768 observations and 9 columns.

I have used Pandas Profiling for Exploratory Data analysis[EDA], Pandas profiling is an efficient way to get an overall as well as in-depth information about the dataset and the variables in it.

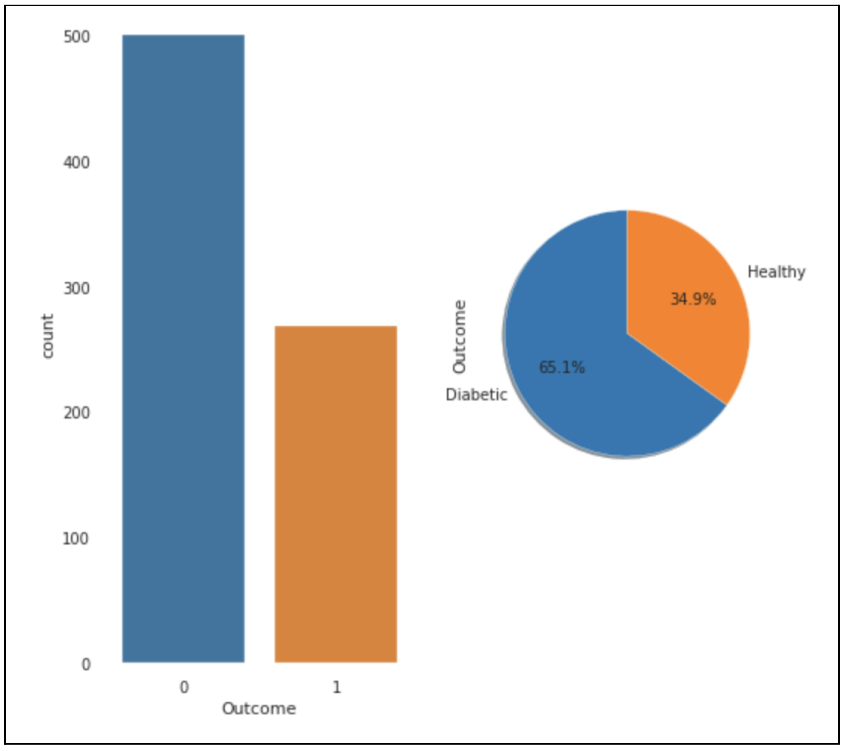
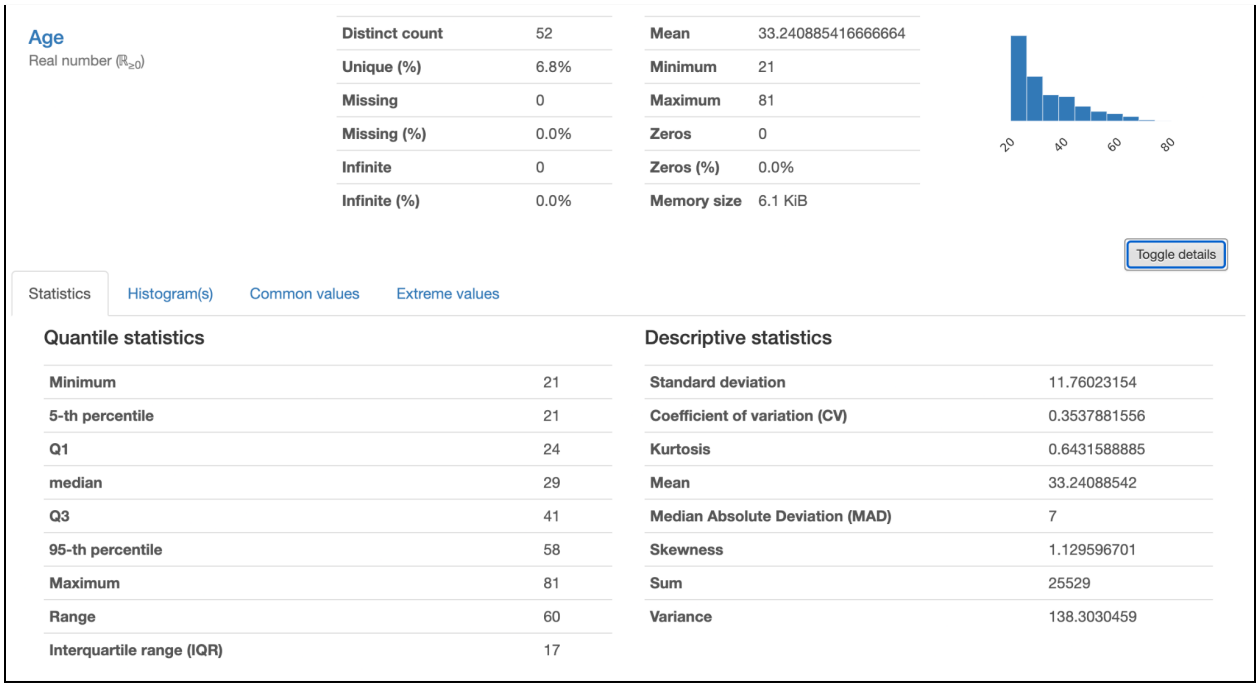


Fig 1: Diabetic VS Healthy

The Above figure shows that in the dataset there are 65.1% diabetic people while the other 34.9 are healthy. Since this result was an outcome based on the 8 features, I have further analysed each attribute in detail for doing feature selection.

Age:



Age
Real number ($\mathbb{R}_{\geq 0}$)

Distinct count	52	Mean	33.240885416666664
Unique (%)	6.8%	Minimum	21
Missing	0	Maximum	81
Missing (%)	0.0%	Zeros	0
Infinite	0	Zeros (%)	0.0%
Infinite (%)	0.0%	Memory size	6.1 KiB

Toggle details

Statistics | Histogram(s) | Common values | Extreme values

Quantile statistics

Minimum	21
5-th percentile	21
Q1	24
median	29
Q3	41
95-th percentile	58
Maximum	81
Range	60
Interquartile range (IQR)	17

Descriptive statistics

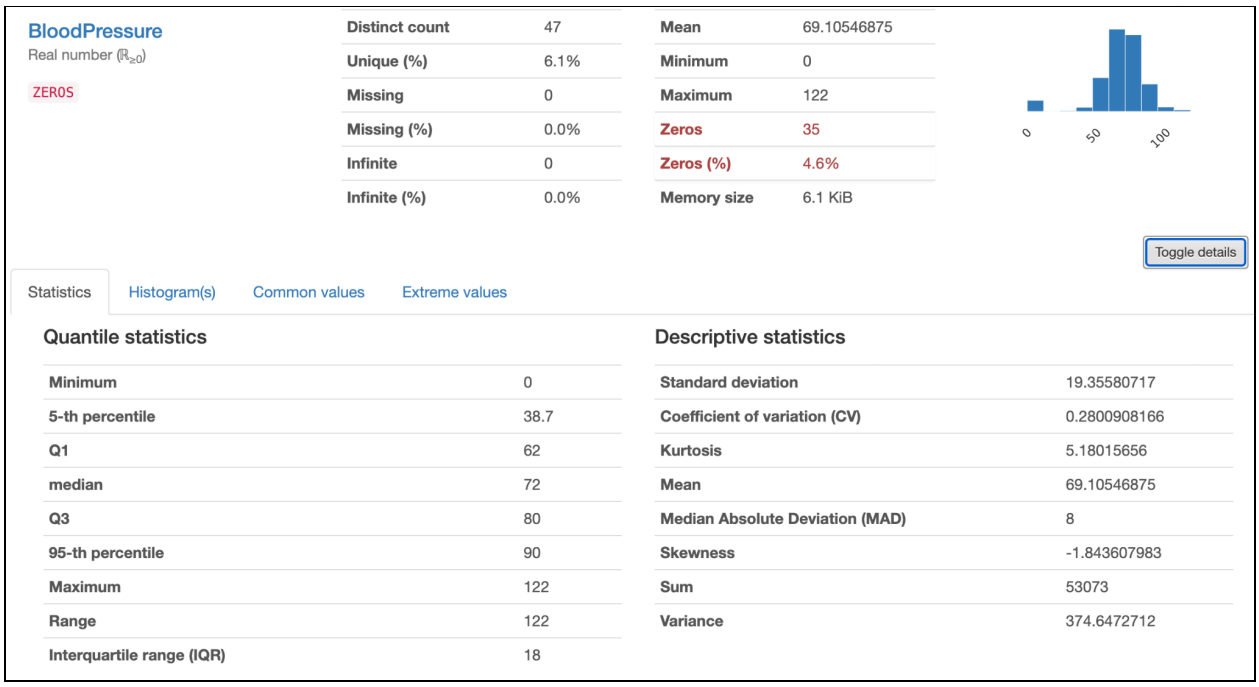
Standard deviation	11.76023154
Coefficient of variation (CV)	0.3537881556
Kurtosis	0.6431588885
Mean	33.24088542
Median Absolute Deviation (MAD)	7
Skewness	1.129596701
Sum	25529
Variance	138.3030459

Fig 2: Independent Variable - Age

- The variable does not follow a normal distribution as it is skewed to the right . The average age is 33 whereas the median age is 29. This says that the analysis is correct since in case of normal distribution, the mean should be approximately equal to the median.

- The minimum age of 21 and the maximum age of 81 says that all the values are normal and there is no noisy data.

Blood pressure:



BloodPressure
Real number (ℝ≥0)
ZEROS

Distinct count	47	Mean	69.10546875
Unique (%)	6.1%	Minimum	0
Missing	0	Maximum	122
Missing (%)	0.0%	Zeros	35
Infinite	0	Zeros (%)	4.6%
Infinite (%)	0.0%	Memory size	6.1 KiB

Toggle details

Statistics | Histogram(s) | Common values | Extreme values

Quantile statistics

Minimum	0
5-th percentile	38.7
Q1	62
median	72
Q3	80
95-th percentile	90
Maximum	122
Range	122
Interquartile range (IQR)	18

Descriptive statistics

Standard deviation	19.35580717
Coefficient of variation (CV)	0.2800908166
Kurtosis	5.18015656
Mean	69.10546875
Median Absolute Deviation (MAD)	8
Skewness	-1.843607983
Sum	53073
Variance	374.6472712

Fig 3: Independent Variable - Blood Pressure

- From the above figure I observed the minimum value for Blood Pressure is 0 (which is not possible). Therefore I replaced such values with median.

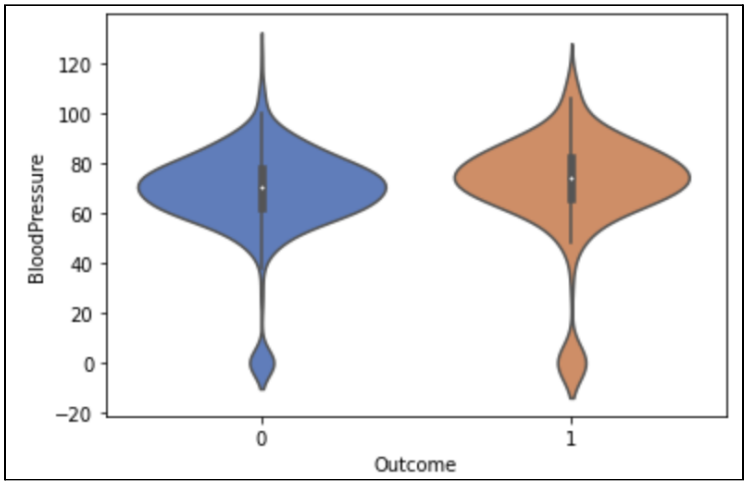


Fig 4: Violin Plot for Blood pressure

- *The above graph is a violin plot for Blood Pressure from which I could see that the box plot for 1 (Diabetic) inside the violin is a little more away from the horizontal axis than the box plot for 0 (Non Diabetic), Which implied that diabetics seem to have a higher blood pressure than the*

non-diabetics. And also that I need to replace all the values that are at the bottom tail of the violin indicating 0 BP.

```
#Replacing the zero-values for Blood Pressure

df1 = diabetesDF.loc[diabetesDF['Outcome'] == 1]
df2 = diabetesDF.loc[diabetesDF['Outcome'] == 0]
df1 = df1.replace({'BloodPressure':0}, np.median(df1['BloodPressure']))
df2 = df2.replace({'BloodPressure':0}, np.median(df2['BloodPressure']))
dataframe = [df1, df2]
diabetesDF = pd.concat(dataframe)
```

Fig 5: Replacing zeros for Blood Pressure

BMI:

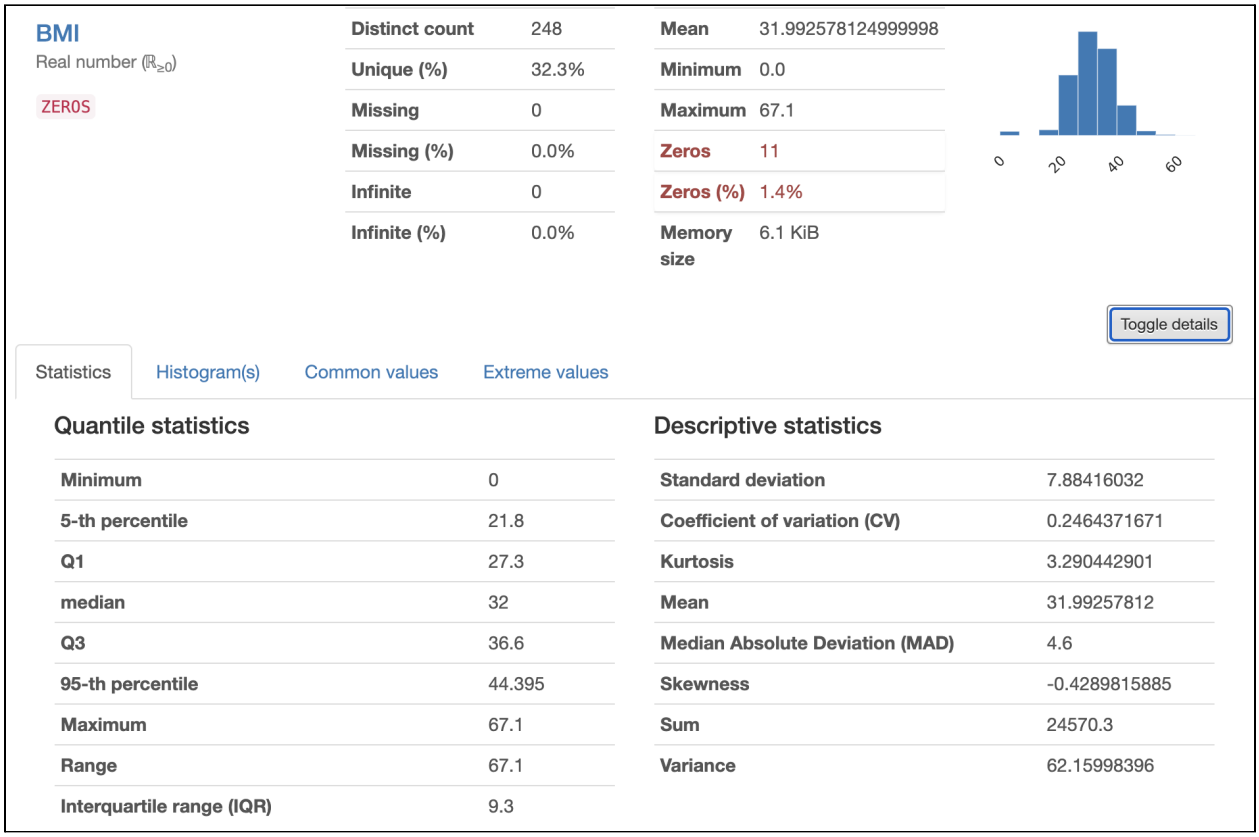


Fig 6: Independent Variable - BMI

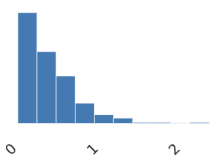
- BMI appeared to be closely following the normal distribution as the mean and median are approximately equal. But It also has a minimum value as 0.
- The violin plot also shows that BMI for diabetics is more than BMI for non-diabetics

Diabetes Pedigree Function:

DiabetesPedigreeF...
Real number ($\mathbb{R}_{\geq 0}$)

Distinct count	517
Unique (%)	67.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.47187630208333325
Minimum	0.078
Maximum	2.42
Zeros	0
Zeros (%)	0.0%
Memory size	6.1 KiB



Toggle details

Statistics | Histogram(s) | Common values | Extreme values

Quantile statistics

Minimum	0.078
5-th percentile	0.14035
Q1	0.24375
median	0.3725
Q3	0.62625
95-th percentile	1.13285
Maximum	2.42
Range	2.342
Interquartile range (IQR)	0.3825

Descriptive statistics

Standard deviation	0.331328595
Coefficient of variation (CV)	0.7021513764
Kurtosis	5.594953528
Mean	0.4718763021
Median Absolute Deviation (MAD)	0.1675
Skewness	1.919911066
Sum	362.401
Variance	0.1097786379

Fig 7: Independent Variable - Diabetes Pedigree Function

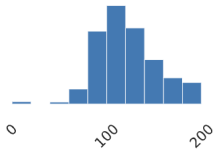
- Diabetes Pedigree Function is a positively skewed variable with no zero values.
- Diabetics seem to have a higher pedigree function than the non-diabetics based on violin plot.

Glucose:

Glucose

Real number ($\mathbb{R}_{\geq 0}$)

Distinct count	136	**Mean**	120.89453125
Unique (%)	17.7%	**Minimum**	0
Missing	0	**Maximum**	199
Missing (%)	0.0%	**Zeros**	5
Infinite	0	**Zeros (%)**	0.7%
Infinite (%)	0.0%	**Memory size**	6.1 KiB



Toggle details

Statistics | Histogram(s) | Common values | Extreme values

Quantile statistics

Minimum	0
5-th percentile	79
Q1	99
median	117
Q3	140.25
95-th percentile	181
Maximum	199
Range	199
Interquartile range (IQR)	41.25

Descriptive statistics

Standard deviation	31.9726182
Coefficient of variation (CV)	0.2644670347
Kurtosis	0.6407798204
Mean	120.8945312
Median Absolute Deviation (MAD)	20
Skewness	0.1737535018
Sum	92847
Variance	1022.248314

Fig 8: Independent Variable - Glucose

- Same as others, Glucose does not follow the normal distribution. We encounter zero-values in this instance as well.
- As well it contains 5 zeros, which is replaced using the same method as earlier

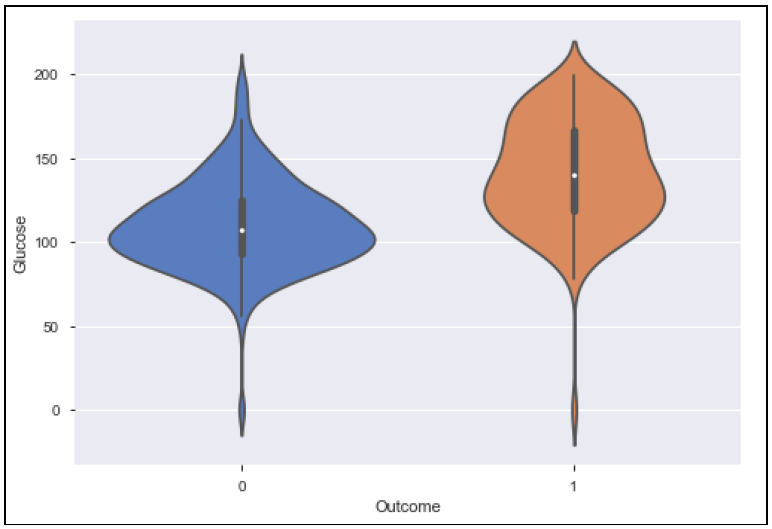


Fig 9: Violin Plot for Glucose

- Violin plot shows a massive vertical distance between the box-plot for Diabetics and Non-Diabetics, indicating that Glucose is a very important variable in model-building.

Insulin:

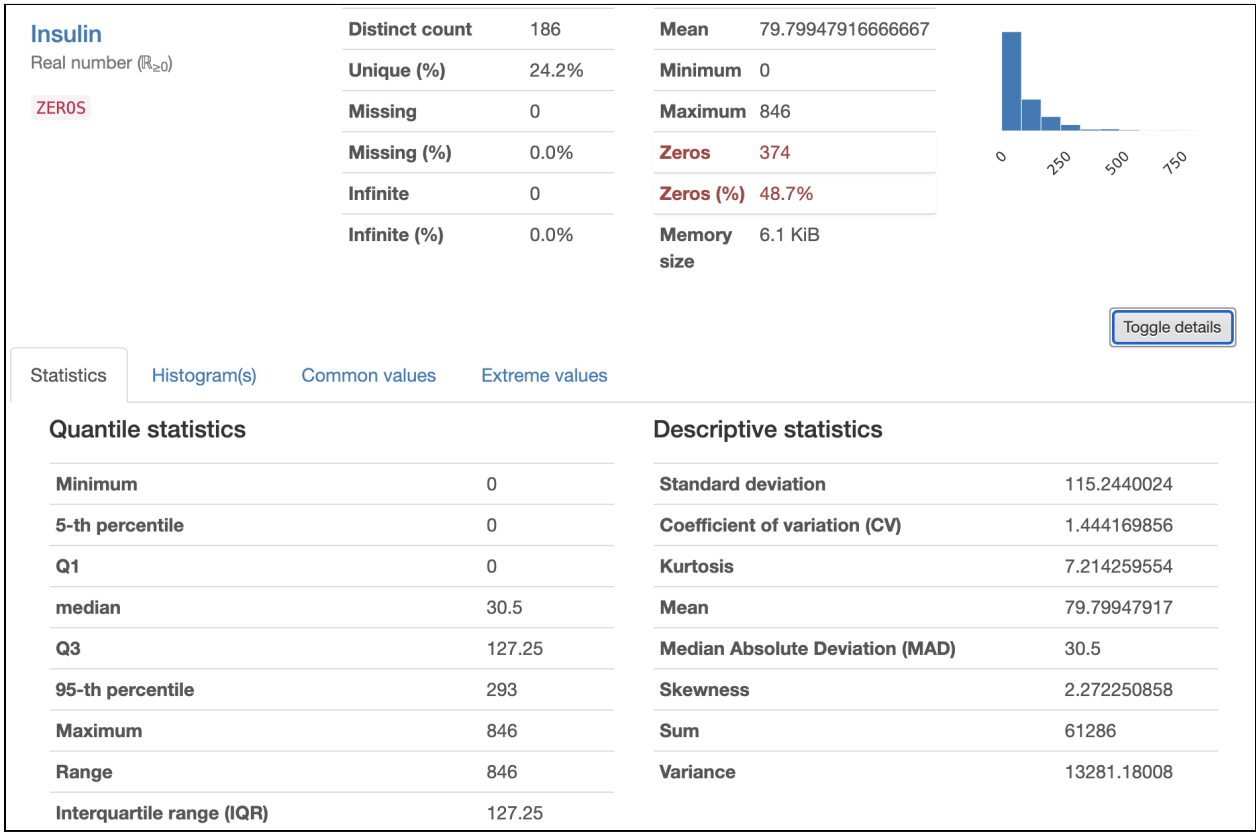
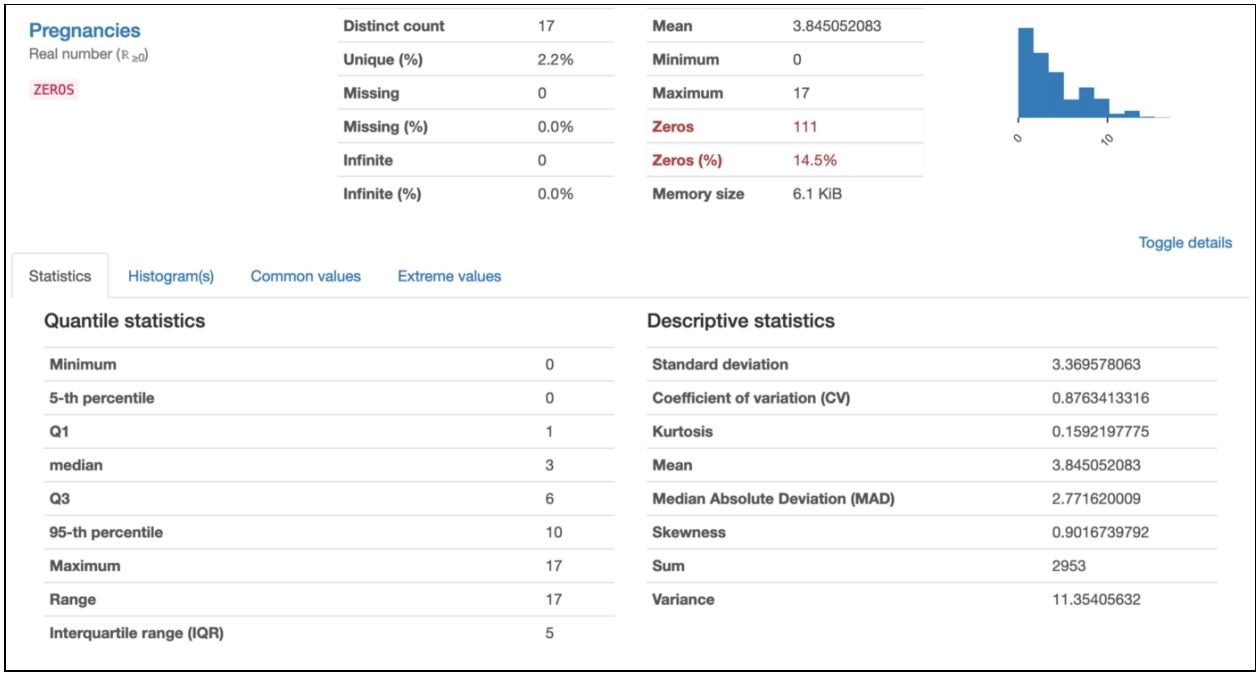


Fig 10: Independent Variable - Insulin

- *As mentioned in the data dictionary available with the dataset, Insulin is the 2-Hour serum insulin (mu U/ml).*
- *The EDA shows the variable is positively skewed.*
- *The occurrence of zero-values is high in this case, making up 48.7% of the data. Hence it has to be replaced.*
- *Violin plot indicates Insulin for Diabetics is lower than Non-Diabetics.*

Pregnancies:



Pregnancies
Real number (ℝ≥0)
ZEROS

Distinct count	17	Mean	3.845052083
Unique (%)	2.2%	Minimum	0
Missing	0	Maximum	17
Missing (%)	0.0%	Zeros	111
Infinite	0	Zeros (%)	14.5%
Infinite (%)	0.0%	Memory size	6.1 KiB

Toggle details

Statistics | Histogram(s) | Common values | Extreme values

Quantile statistics

Minimum	0
5-th percentile	0
Q1	1
median	3
Q3	6
95-th percentile	10
Maximum	17
Range	17
Interquartile range (IQR)	5

Descriptive statistics

Standard deviation	3.369578063
Coefficient of variation (CV)	0.8763413316
Kurtosis	0.1592197775
Mean	3.845052083
Median Absolute Deviation (MAD)	2.771620009
Skewness	0.9016739792
Sum	2953
Variance	11.35405632

Fig 11: Independent Variable - Pregnancies

- The variable is positively skewed with 14.5% zero values.
- However the other 85.5% of Non-Zero values indicate that the study mainly includes female data only.
- Looking at the behavior using violin plots I observed diabetic women had more pregnancies than non-diabetic.

SkinThickness:

- The data is positively skewed with 29.6% of zero values.
- Analysis also shows Skin Thickness for Diabetics is more than that of Non-Diabetics

Analysing Correlation:

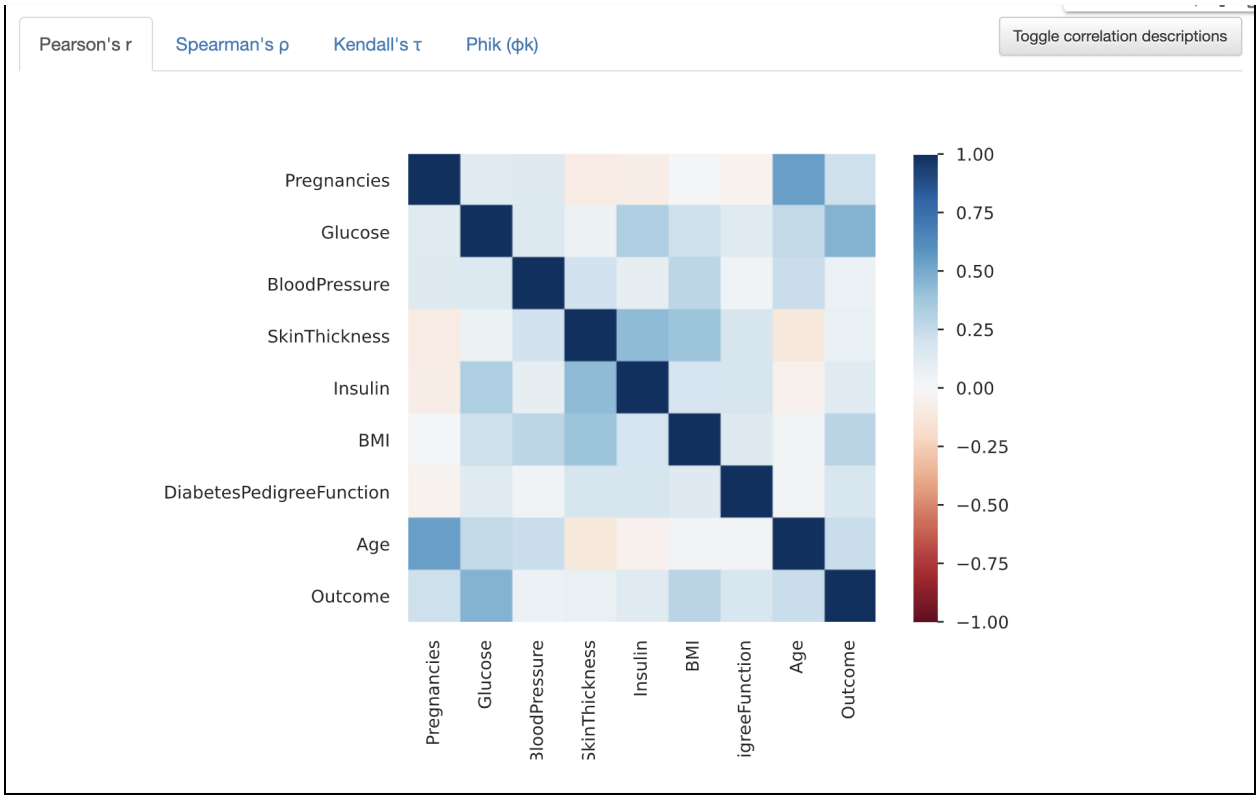


Fig 12: Pearson's Correlation

- Pearson's correlation coefficient illustrates the relationship between variables. From the above figure, a significant correlation can be observed between Pregnancies and Age.

Treating Outliers and Non-Normality:

- After loading the dataset I first converted all the string values to numeric
- Normalized each column to values in the range of 0 to 1

```
[ ]  column=['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']

     #convert all string to float
     for i in column:
       diabetesDF[i] = diabetesDF[i].astype(float)

     # Rescale dataset columns to the range 0-1
     scaler = MinMaxScaler()
     diabetesDF = pd.DataFrame(scaler.fit_transform(diabetesDF), columns=diabetesDF.columns)
     diabetesDF.head()
```

Fig 13: Data Normalization

Data Splitting:

- I have used k-fold cross validation to estimate the performance of the learned model on unseen data.
- I have constructed and evaluated k models and estimated the performance as the mean model performance.
- And used classification accuracy to evaluate each model.

Designing Logistic Regression:

- As per my research Logistic Regression uses an equation as the representation, very much like linear regression. Input values (X) are combined linearly using weights or coefficient values to predict an output value (y).
- A key difference from linear regression is that the output value being modeled is a binary value (0 or 1) rather than a numeric value.

$$yhat = 1.0 / (1.0 + e^{(-(b0 + b1 * x1))})$$

  - Where e is the base of the Euler's number,
  - yhat is the predicted output,
  - b0 is the bias or intercept term
  - b1 is the coefficient for the single input value (x1).

- Each column in the data has been associated with 'b' coefficient that is learned from training data.

Stochastic Gradient Descent:

- As per my research Gradient Descent is the process of minimizing a function by following the gradients of the cost function.
- This involves knowing the form of the cost as well as the derivative so that from a given point you know the gradient and can move in that direction.

- I have used stochastic gradient descent to minimize the error of the model on training data since it updates the coefficients every iteration .
- So the way my model works is that each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated and the model is updated in order to reduce the error for the next prediction.
- After each iteration, the coefficients (b) is updated

```
# Estimate logistic regression coefficients using stochastic gradient descent
def stochastic_gradient_descent(train, learning_rate, epochs):
  Coeff = [0.0 for i in range(len(train.iloc[0]))]
  for epoch in range(epochs):
    sum_of_error = 0
    for (idx, rows) in train.iterrows():
      row=[rows.Pregnancies,rows.Glucose,rows.BloodPressure,rows.SkinThickness,rows.Insulin,rows.BMI,rows.DiabetesPedigreeFunction,rows.Age,rows.Outcome]
      y_out = predict(row, Coeff)
      error = row[-1] - y_out
      sum_of_error += error**2
      Coeff[0] = Coeff[0] + learning_rate * error * y_out * (1.0 - y_out)
      for i in range(len(row)-1):
        Coeff[i + 1] = Coeff[i + 1] + learning_rate * error * y_out * (1.0 - y_out) * row[i]
      print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, learning_rate, sum_of_error))
  return Coeff
```

Fig 14: Stochastic Gradient Descent Implementation

- I have used sigmoid function to predict the outcome for a given row of input

```
[11] # Make a prediction with coefficients - Sigmoid Funciton
     def predict(row, coefficients):
       y_out = coefficients[0]
       for i in range(len(row)-1):
         y_out += coefficients[i + 1] * row[i]
       return 1.0 / (1.0 + exp(-y_out))
```

Fig 15: Prediction using Sigmoid Function

Hyperparameters:

- A k value of 5 is used for cross-validation, giving each fold 768/5 = 153.6 or just over 150 records to be evaluated upon each iteration.
- A learning rate of 0.2 and 10000 training epochs are chosen after a little experimentation.

Result:

- Model gives an accuracy of about 77.5%

```
Scores: [73.8562091503268, 78.43137254901961, 81.69934640522875,
75.81699346405229, 75.81699346405229]
Mean Accuracy: 77.124%
```

Fig 16: Model scores & Accuracy

Part 2: Implement Neural Networks

Index

List of Figures

1] Dataset Preprocessing & Splitting:

- I have used tensorflow to split the dataset into 70% training and 30% testing sets.
- Then I have applied one hot encoding on the outcome converting 1 into [1,0] and 0 into [0,1]

```
label = []
for lab in df_label:
    if lab == 1:
        label.append([1, 0])  # class 1
    elif lab == 0:
        label.append([0, 1])  # class 0
```

Fig 17: Data Preprocessing

- Then finally converted all the input data into numpy array to feed in to the model for training

2] Neural Network Architecture:

```
Input Layer: 500 nodes - Sigmoid Activation, L2 Regularization with
Learning Rate(lr) of 0.00001
Dropout Layer 1: Working at the rate of 0.2
Hidden Layer 1: 100 nodes - Sigmoid Activation, L2 Regularization with
Learning Rate(lr) of 0.00001
Dropout Layer 2: Working at the rate of 0.2
Output Layer: 2 node - Softmax activation, L2 Regularization with
Learning Rate(lr) of 0.00001
```

This Neural Network is feedforward and fully connected. Every node in a given layer will be connected to all nodes in the next layer.

- **Activation Function:**
  - The Sigmoid activation function has been used for all nodes except those in the output layer, which is yielding either 1 or 0.
  - The output layer uses Softmax activation to yield a value between 0 to 1, which would be the predicted chance of diagnosis.

- **Loss or Cost Function:**
  - Mean Squared Error loss function has been used because the output contains either 1 for positive diagnosis or 0 for negative diagnosis. MSE measures the average of the squares of the errors, that is the average squared difference between the estimated values and the actual value.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE	=	mean squared error
n	=	number of data points
Yi	=	observed values
Y^i	=	predicted values

- **Optimizer:**
  - Adam has been used as the optimizer because it is an efficient way to apply gradient descent. Gradient descent is an optimization algorithm used to find the values of the parameters of a function that minimizes a loss function, in this case the mean squared error.
  - Learning Rate is 0.00001

- **Metrics:**
  - Tensorflow accuracy metrics is used to evaluate the model

- **Model:**
  - The model is trained for 5000 epochs
  - With a batch size of 10

- **Result & Comparison:**

```
Evaluate on test data
2/2 [==============================] - 0s 5ms/step - loss: 0.1698 -
accuracy: 0.7806
Accuracy : 78.05627679824829 %
```

Fig 18: Model Loss & Accuracy

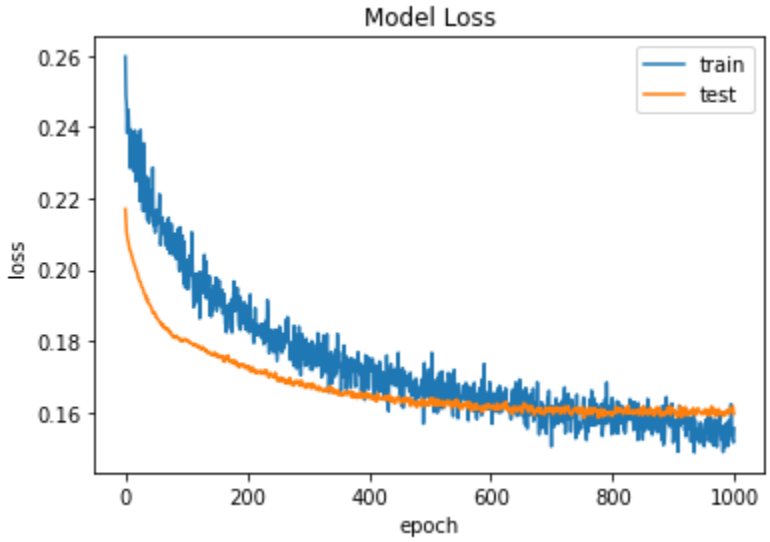


Fig 19: Model Loss & Epoch

Network Architecture	L1 regularization % Accuracy	L2 regularization Accuracy%	Dropout %Accuracy
10-12-2	69.9878%	65.9667%	64.9982%
8-12-2	70.0574%	63.8747%	68.0781%
25-10-2	49.1345%	66.9447%	62.1034%
500-100-2	72.3649%	75.0667%	78.7803%
80-400-2	59.6473%	71.6911%	65.8867%
300-120-2	65.1278%	68.0778%	73.4623%
100-300-2	73.7618%	63.0643%	59.9867%

Fig 20: Regularization methods Comparison table